Object Model + Circuits Integration

- Author: @ Rob Giusti
- Sign Off:
- PR Review: @ Tyler Leonhardt @ William Chilcote
- Design Doc: OM Rules Complete Edition v5
- Test Plan:
- Change Log: First Draft, 6/14/22

Table of Contents

- 1 Technical Goals
- 2 Detailed Design
- 2.2 Object Model Circuit Scopes
- 3 Unresolved Questions
- 4 See Also
- 5 Frequently Asked Questions
- 6 Appendices

1 Technical Goals

1. OM Objects can be created with Object Boards
    a. OM Objects can have their Object Boards toggled on and off according to the same specifications as Deletable Object Boards
2. Circuits Scopes are aligned to OM Scopes, specified by OM Rules Complete Edition v5
3. Circuit objects in existing rooms can be migrated into an OM room with automatic conversions

2 Detailed Design

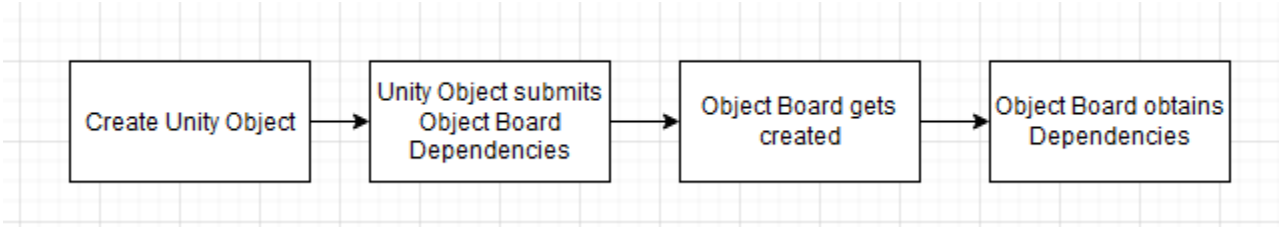
The work for this project largely falls into two buckets, along the first two technical goals: **OM Object Boards** and **OM Circuit Scopes**

2.1 Object Model Object Boards

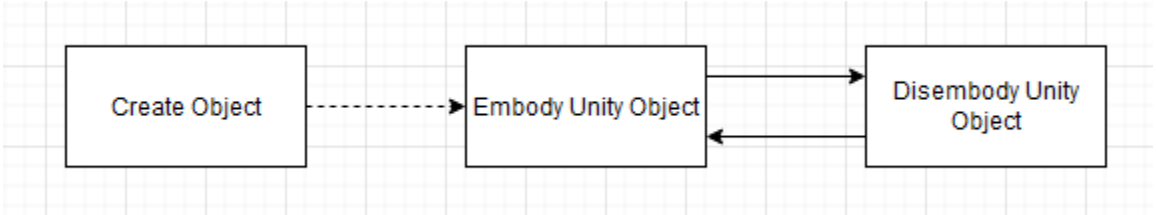
OM greatly changes the creation flow for objects with object boards, due to the addition of the “embodiment” process, as well as changes to prefab and creation flow.

2.1.1 Current State

Currently, Object Boards are created immediately after a requesting Unity Object, and obtain an `IControlPanelSelf` component from the object that it is on.


Create Unity Object
Unity Object submits Object Board Dependencies
Object Board gets created
Object Board obtains Dependencies

The Object Model system handles objects differently in that an object is not always “embodied” into an Object View / Unity Object. First, an Object is created, and then (later or immediately), it can embodied into a Unity Object, and can then be “disembodied” from that object.



Because objects can exist in a “disembodied” state, we should no longer rely entirely on Unity `MonoBehaviours` to supply these dependencies.

OM's flows for creating objects are different from the `CreationManager` flows that are used for creation today, and currently lack much of the functionality that we need in order to get Object Boards created on the creation of an appropriate object and then associated with the correct Object View.

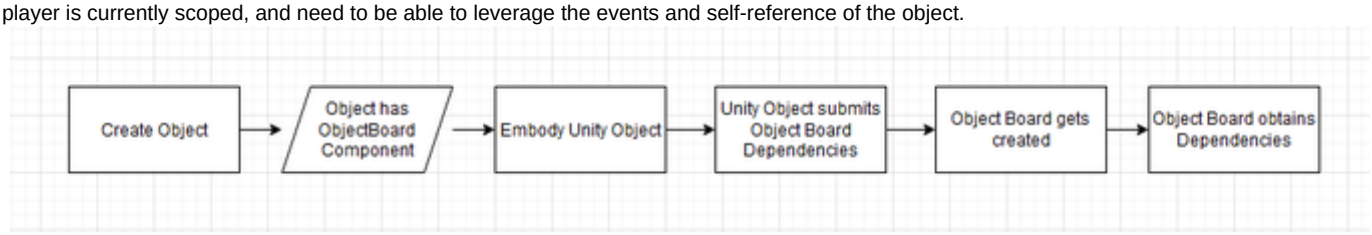
2.1.2 General Plan

On the OM data side, we need to be able to persist information about Object Boards in an OM component: `ObjectBoardData` (which can be found in Appendix 6.1)

Objects that, by default, should be created with Object Boards, should have empty `ObjectBoardData` components in their Object Prefab Archetypes.

2.1.3 Short-Term Plan

In the short term, we can keep the existing infrastructure for dependency handling, however it will require any Object with an Object Board to be immediately embodied upon creation, and to stay embodied. This is because Circuits logic can be run in any scope, regardless of where the player is currently scoped, and need to be able to leverage the events and self-reference of the object.

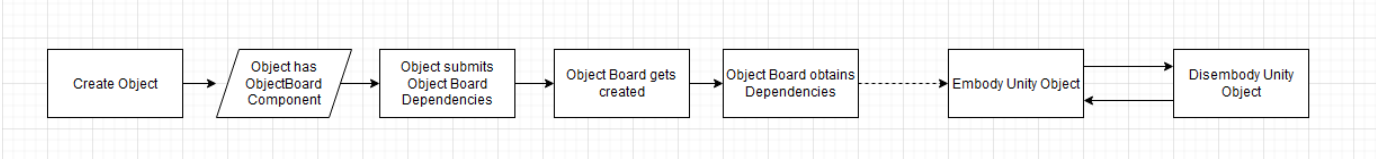


In this model, we rely on many of the existing code paths: The dependencies still get fulfilled by a `ControlPanelCircuitObjectBase` component on the Unity Object that is created before the Object Board, and the data from the `ObjectBoardData` is packed into the `instantiationParams` passed into `PhotonNetwork.InstantiateEx` so that the Photon instantiation flow still has the correct information where it expects.

In @ Rob Giusti 's exploratory work, there is a proof-of-concept of this flow working for [OMShapeContainer].

2.1.4 Long-Term Plan

In the longer term, we need a strategy that will allow us to disembody circuit-enabled objects while still fulfilling their circuits dependencies. This will be done by migrating data and logic out of `MonoBehaviour` components, and into Object Components and Component System Jobs.



This is a bit beyond the scope of this core integration work, since this requires work to port each individual logical gadget, but should be kept in mind as a north-ish star for what we're working towards.

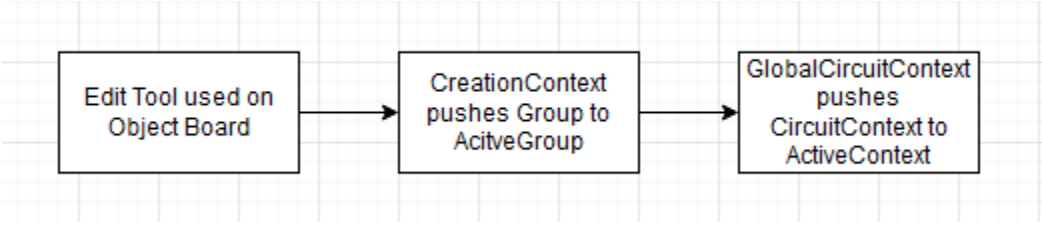
In this model, we will need a non-`MonoBehaviour` implementation of `IControlPanelDependencies` that can be submitted to the Circuits system, and fulfill the requirements there out of just Object Components.

The first candidates for this transformation will be the Piston and Rotator gizmos. Note that we have *just nearly* caught up to porting all the dynamic props and components from CV1 to CV2, and this will unfortunately introduce another long-term migratory workstream.

2.2 Object Model Circuit Scopes

2.2.1 Current State

Currently, `Chip.ChipCreationContextGroup` is an `ICreationContextGroup` that gets put onto a stack of `ActiveGroups` in the `CreationContext` when using the Edit Tool on a `CircuitGraphNode`.

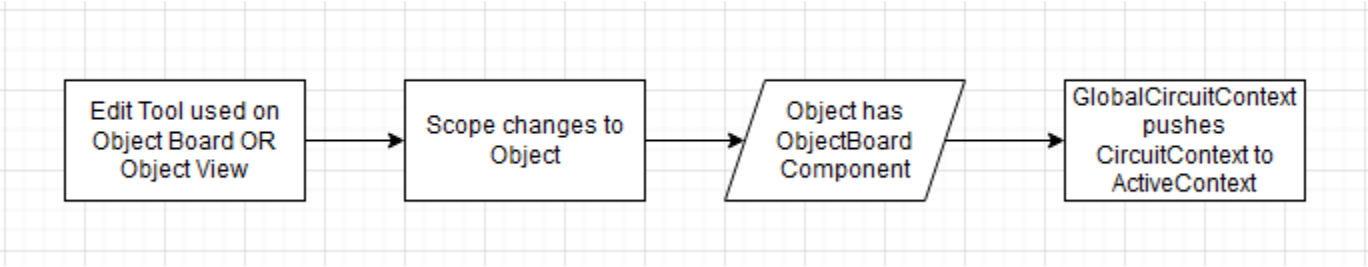

Edit Tool used on Object Board
CreationContext pushes Group to AcitveGroup
GlobalCircuitContext pushes CircuitContext to ActiveContext

The concept of `ICreationContextGroup` is that each are a group of objects that, once you push onto the `ActiveGroup` stack in your `CreationContext`, you are then editing that group of objects instead of normal Rec Room Objects in the scene. Aside from Circuits, this is also used for `CurveControlPointGroup`, `AnimationGroup`, as well as CV1 circuit boards.

2.2.2 Migrating CircuitGraph from Groups to Scopes

With Object Model's rules for Circuit scoping, we can no longer have Object Boards use the Grouping system in `CreationContext`, because we need circuits and objects to both be editable at the same time.

Using the edit tool on an OM Container with an Object Board or on the Object Board will change the OM Scope to be the OM Container, and then while in that Scope, the proper `CircuitGraph` will be pushed to the `GlobalCircuitGraph`, aligning the scopes of the two systems.


Edit Tool used on Object Board OR Object View
Scope changes to Object
Object has ObjectBoard Component
GlobalCircuitContext pushes CircuitContext to ActiveContext

This will require writing some extra logic for the `Chip` class, such that handles world fade and selection filtering differently, since it will no longer be able to rely on the Group paradigm to ensure that only chips in the current graph can be interacted with and fully visible.

Because of Deletable Circuit Boards, `GlobalCircuitGraph` also now needs to have a concept of its active edit graph being “none.” In addition, we need to have some extra logic for creating/adding object boards into scopes that do not already have object boards, as well as some extra UX for warning players about removing object boards that have nested boards within them.

3 Unresolved Questions

1. What is the fate of Circuit Board in the OM world?
   a. Circuit Boards do not have an object in the scene/hierarchy that they are tethered to, they have nothing that the OM system can properly scope into, which would mean that CircuitBoards can't be migrated off of being `ICreationContextGroup` and leveraging the old UX pattern.
   b. However, there is little difference between Circuit Boards and an empty container with an Object Board; if we were to turn all CBs into empty containers with OBs as part of the upgrade flow for OM, it would alleviate the need to maintain two different UX systems for interacting with graph nodes.
2. How is Object Model planning on spawning OM Objects for prefabs using `SpawnableToolType` spawning methods?
   a. Currently, the plan in Short-Term Plan relies on Object Model versions of current Rec Room prefabs (for example, a Button).
   b. Exploratory work has a version of this working for [OMShapeContainer], which already uses its own prefab rather than the default [ObjectView] prefab, but has a different code path that may not be permanent.
3. How do Saving/Loading work with OM objects and their properties right now?
   a. We were already serializing GraphId
4. How do we version OM data if we make versioning changes to circuits (ie. changing Guids of Graphs during a

4 See Also

1. OM Rules Complete Edition v5: Circuits
2. Pull Request 18692: Nested Circuit Boards
3. Pull Request 19314: Deletable Object Boards

5 Frequently Asked Questions

6 Appendices

6.1 New OM Component Data

```
[OMComponentData]
public struct ObjectBoardData : IComponentData
{
    /// <summary>
    /// The inner graph Id for this Object's Object Board. If this
object doesn't
    /// have an Object Board, this will be equal to Guid.Empty
    /// </summary>
    [OMProp(1)]
    public SerializableGuid graphId;

    /// <summary>
    /// Flag for if this Object needs to have an Object Board created
    /// at the next opportunity
    /// </summary>
    [OMProp(2)]
    public bool needsObjectBoard;
}
```

Common Appendices

Summary

! *Must include if no design doc exists.*

A summary of the design.

Milestones

! *Must include if the development time forecast is greater than 4 weeks.*

Individual shippable parts that this design may be broken into or reasons it cannot be.

Drawbacks

! *Must include if an Alternatives sections is included.*

Drawbacks of the design.

Alternatives

! *Must include if alternative designs are proposed.*

Alternative designs which attempt to accomplish the same goals.

Rollout Plan

! *Consider if there is a phased release. Pairs well with Milestones or Beta section.*

How this design will be made available to the public.

General Appendices

Testability

! *Consider for larger or foundational work.*

How the design is made accessible from unit tests or automated tests.

Debugability

❕ *Consider if new `LogFlags` or other debugging capabilities are changed or added.*

Changes to debugging capabilities.

Server Game Configs

❗ *Must include if GameConfigs are changed or added.*

Changes to GameConfigs.

Assembly Changes

❗ *Must include if .asmdef files are added or changed.*

Changes to .asmdef files.

Workflow Updates

❗ *Must include if there are changes to developer or designer workflows.*

New things folks have to do with tools because of this design.

Server

❗ *Must include if this design requires server work.*

Changes to the server required for this design.

UI

❗ *Must include if this design includes non-generated UI.*

UI page or system changes.

Assets

❕ *Consider if this design requires significant assets.*

Assets required for this design.

New Plugins and Packages

❗ *Must include if plugins and packages are added or changed.*

Plugins and package modifications.

Performance

❗ *Must include if performance is significantly impacted.*

Performance implications.

Loading Time

❗ *Must include if loading experiences a > 1 second change.*

How loading time will be impacted. Could be worsened or improved.

Frame-Rate

❗ *Must include if frame-rate is impacted by > 1 FPS on any device.*

How frame-rate will be impacted. Could be worsened or improved.

Memory

❗ *Must include if memory is impacted by > 1 MB on any device.*

How memory will be impacted. Could be worsened or improved.

Download Size

❗ *Must include if download size is impacted by > 10 MB on any device.*

How download size will be impacted. Could be worsened or improved.

Analytics Appendices

Amplitude

❗ *Must include if new data is captured in Amplitude.*

New Amplitude events and properties.

Databricks

❗ *Must include if new data is captured in Databricks.*

New Databricks data.

Statsig Split Tests

❗ *Must include if the change includes Statsig split tests.*

Creation Appendices

Protobufs

❗ *Must include if any .proto file changes besides action_data.proto.*

Changes to protobuf files.

Versioning

❗ *Must include if a version is incremented for any component of the game.*

Why existing content is migrated and how.

Beta Plan

❗ *Must include if beta content is added.*

Why the design is beta and the criteria for exiting beta.

Circuit Actions

❗ *Must include if action_data.proto, CircuitRoot.RootReduce, or CircuitRoot.RootRequest are changed.*

Changes made to action_data.proto, `CircuitRoot.RootReduce`*, and* `CircuitRoot.RootRequest`*.*

Circuit Types

❗ *Must include if a new class or static member is added that inherits from* `CircuitType`.

New `CircuitTypes` *and their backing data.*

New Global Circuit Events

❗ *Must include if* `BuiltInEvents` changes.

New global circuit events and an explanation about why they can't be local events.

Creation Resource Requirements

❗ *Must include if creation resources will be impacted. This includes new spawnables and chips.*

How this design impacts creation resources.

Logic CPU Heat

❗ *Must include if CPU heat is impacted.*

How much CPU heat this design requires.

Logic Network Heat

❗ *Must include if network heat is impacted.*

How much network heat this design requires.

Logic Memory

❗ *Must include if some memory limit must be in place.*

How much network heat this design requires.

Ink

❗ *Must include if the design has a non-standard ink cost.*

How much ink this design requires.

Throttling

❗ *Must include if the design is throttled in ways not captured by creation resources.*

How this design is throttled outside of standard creation resources.

Other

❕ *Consider if creation resources are impacted in other ways. Like spawn limits.*

How this design impacts creation resources in non-standard ways.

Multiplayer Appendices

Photon RPCs

❗ *Must include if a* `[PunRPC]` *changes or if new ones are added.*

Changes to `[PunRPC]`*s.*

Photon Custom Properties and Synchronized Fields

❗ *Must include if a Photon custom property or* `SynchronizedField` *changes or if new ones are added.*

Changes to Photon custom properties and `SynchronizedFields`*.*

Multiplayer

❕ *Consider if there are multiplayer changes not captured by other appendices.*

General changes to multiplayer.